Overfitting and Underfitting

- Did you know that there’s one mistake…
- …that thousands of data science beginners unknowingly commit?
- And that this mistake can single-handedly ruin your machine learning model?
- No, that’s not an exaggeration. We’re talking about one of the trickiest obstacles in applied machine learning: *overfitting*.

Example

- Let's say we want to predict if a student will land a job interview based on her resume.
- Now, assume we train a model from a dataset of 10,000 resumes and their outcomes.
- Next, we try the model out on the original dataset, and it predicts outcomes with 99% accuracy… wow!
- But now comes the bad news.
- When we run the model on a new ("unseen") dataset of resumes, we only get 50% accuracy… uh-oh!
- **Our model doesn't *generalize* well from our training data to unseen data.**
- This is known as overfitting, and it's a common problem in machine learning and data science.

Signal vs. Noise

- In predictive modeling, you can think of the “signal” as the true underlying pattern that you wish to learn from the data.
- “Noise,” on the other hand, refers to the irrelevant information or randomness in a dataset.
- For example, let’s say you’re modeling height vs. age in children. If you sample a large portion of the population, you’d find a pretty clear relationship:
- This is the signal.
- However, if you could only sample one local school, the relationship might be muddier. It would be affected by outliers (e.g. kid whose dad is an NBA player) and randomness (e.g. kids who hit puberty at different ages).

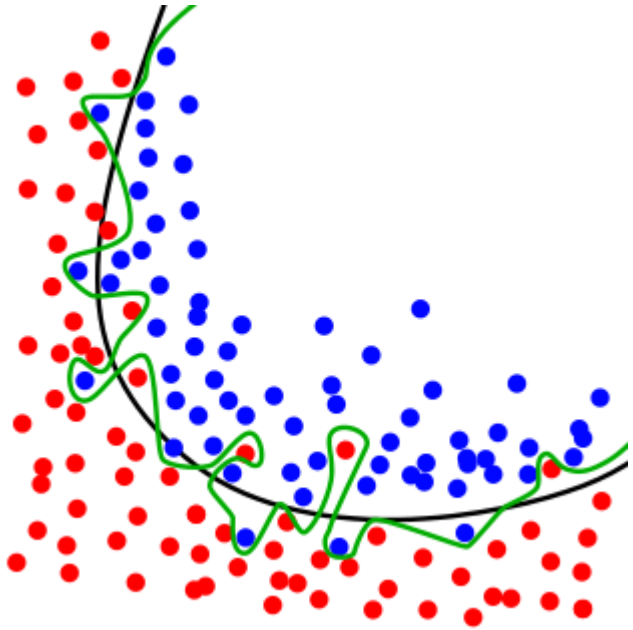
Noise interferes with signal.

- Here's where machine learning comes in. A well functioning ML algorithm will separate the signal from the noise.
- If the algorithm is too complex or flexible (e.g. it has too many input features or it's not properly regularized), it can end up "memorizing the noise" instead of finding the signal.
- This overfit model will then make predictions based on that noise. It will perform unusually well on its training data… yet very poorly on new, unseen data.

Goodness of Fit

In statistics, *goodness of fit* refers to how closely a model's predicted values match the observed (true) values.

A model that has learned the noise instead of the signal is considered "overfit" because it fits the training dataset but has poor fit with new datasets.



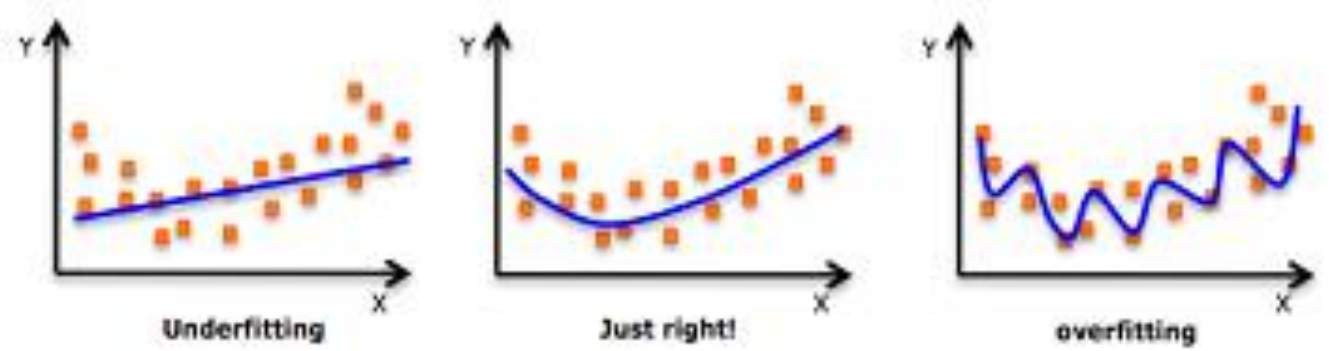
While the black line fits the data well, the green line is overfit.

Overfitting

- Overfitting means that model we trained has trained “too well” and is now, well, fit too closely to the training dataset.
- This usually happens when the model is too complex (i.e. too many features/variables compared to the number of observations).
- This model will be very accurate on the training data but will probably be very not accurate on untrained or new data. It is because this model is not generalized (or not AS generalized), meaning you can generalize the results and can’t make any inferences on other data, which is, ultimately, what you are trying to do.
- Basically, when this happens, the model learns or describes the “noise” in the training data instead of the actual relationships between variables in the data. This noise, obviously, isn’t part in of any new dataset, and cannot be applied to it.

Underfitting

- In contrast to overfitting, when a model is underfitted, it means that the model does not fit the training data and therefore misses the trends in the data.
- It also means the model cannot be generalized to new data. This is usually the result of a very simple model which has not enough predictors/independent variables.
- It could also happen when, for example, we fit a linear model (like linear regression) to data that is not linear. This model will have poor predictive ability (on training data and can't be generalized to other data).
- Train/test split and cross validation are two rather important concepts in data science and data analysis used as tools to prevent (or at least minimize) overfitting and underfitting.
- But the tools help to avoid overfitting more than underfitting.



Underfitting

- Underfitting occurs when a model is too simple – informed by too few features or regularized too much – which makes it inflexible in learning from the dataset.
- Simple learners tend to have less variance in their predictions but more bias towards wrong outcomes.
- On the other hand, complex learners tend to have more variance in their predictions.
- **Both bias and variance are forms of prediction error in machine learning.**

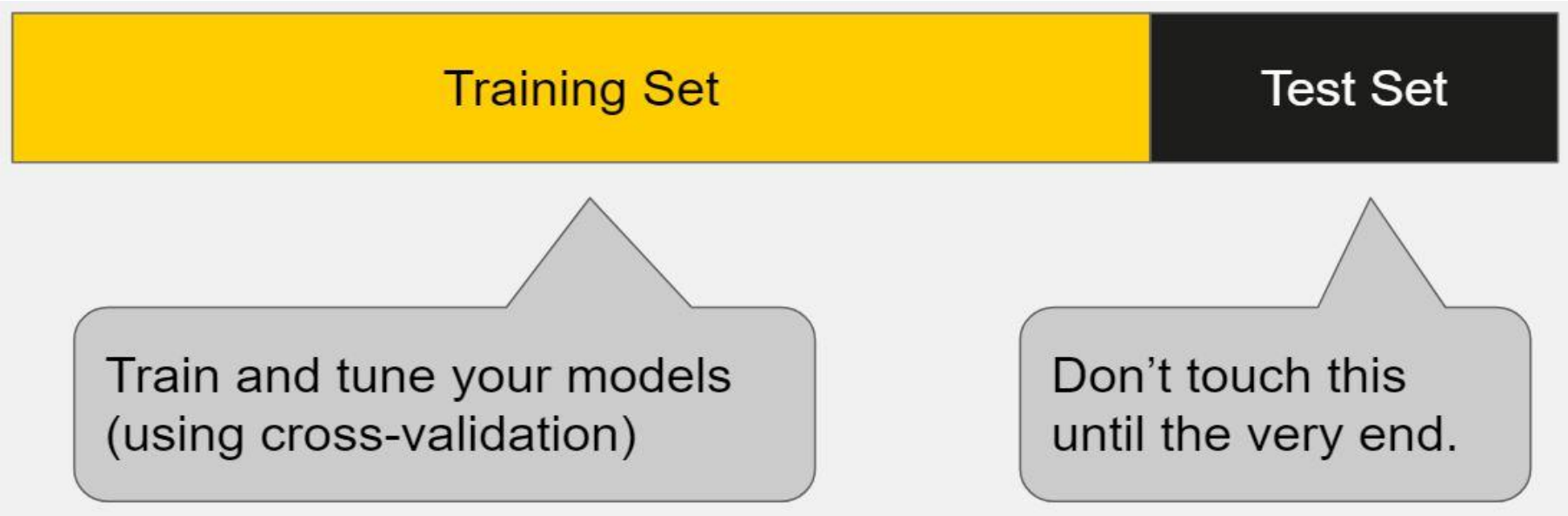
Underfitting

- Typically, we can reduce error from bias but might increase error from variance as a result, or vice versa

How to Detect Overfitting

A key challenge with overfitting, and with machine learning in general, is that we can't know how well our model will perform on new data until we actually test it.

To address this, we can split our initial dataset into separate *training* and *test* subsets.

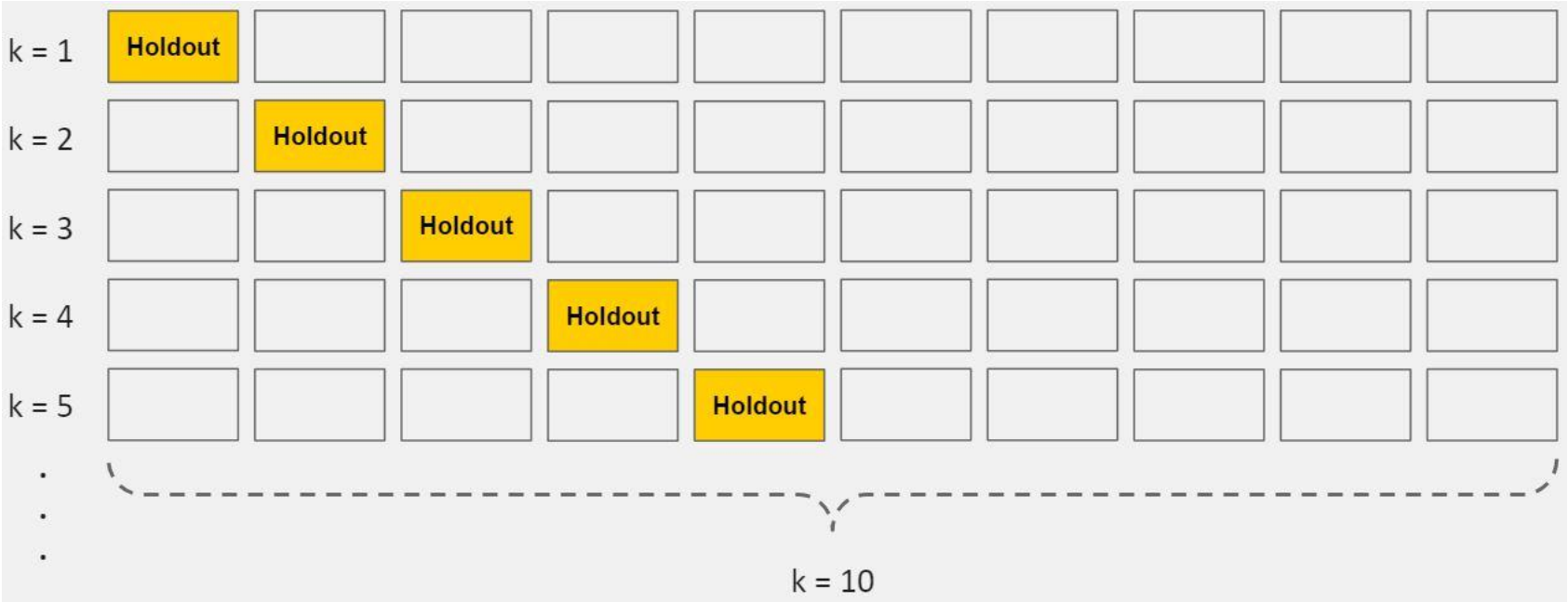


If our model does much better on the training set than on the test set, then we're likely overfitting.

Cross-validation

- Cross-validation is a powerful preventative measure against overfitting.
- The idea is clever: Use your initial training data to generate multiple mini train-test splits. Use these splits to tune your model.

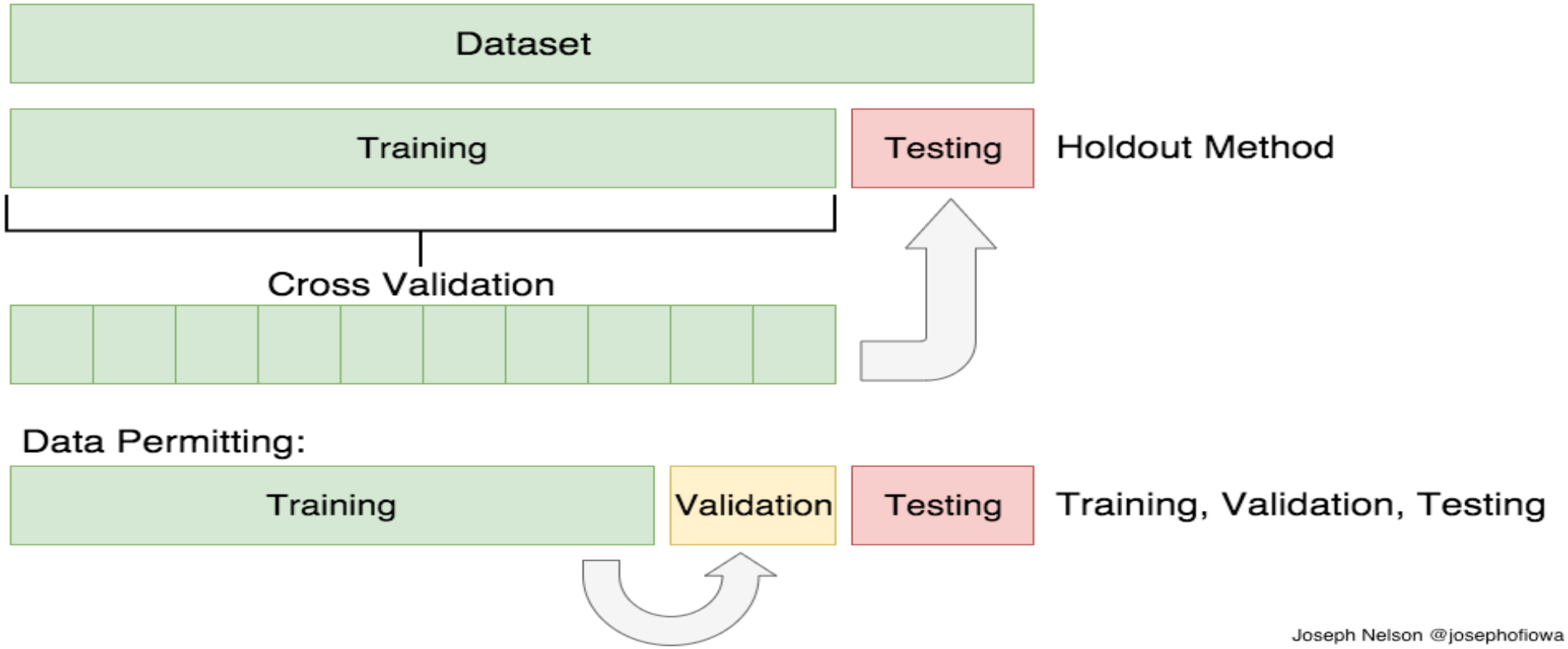
K-Fold Cross-Validation



Cross-validation allows you to tune hyperparameters with only your original training set. This allows you to keep your test set as a truly unseen dataset for selecting your final model.

Cross Validation

- Meaning, we split our data into k subsets, and train on k-1 one of those subset. What we do is to hold the last subset for test. We're able to do it for each of the subsets.



Cross Validation

- There are a bunch of cross validation methods:
- **K-Folds Cross Validation**
- **Leave One Out Cross Validation** (LOOCV).

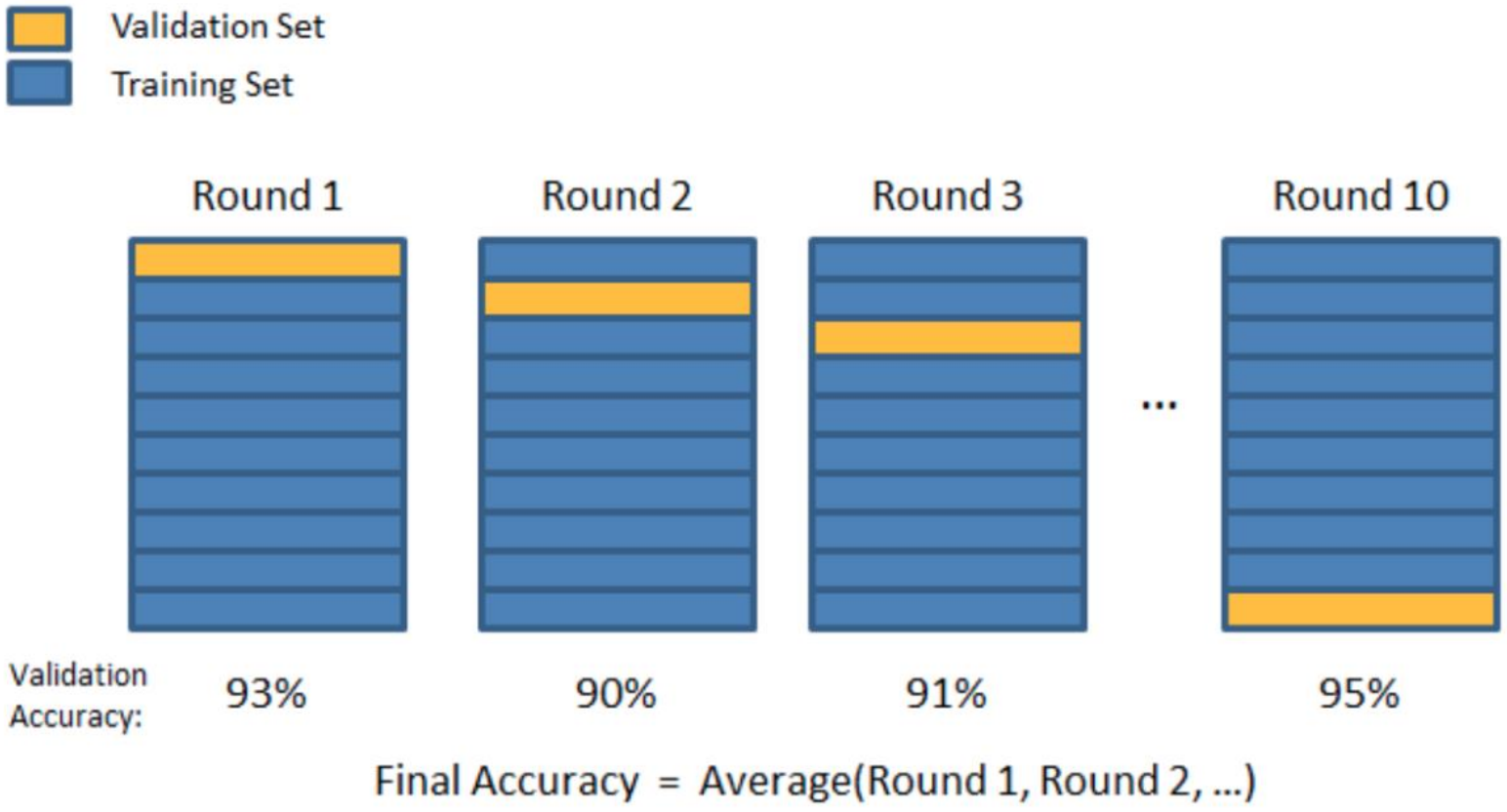
In K-Folds Cross Validation we split our data into k different subsets (or folds).

We use k-1 subsets to train our data and leave the last subset (or the last fold) as test data.

We then average the model against each of the folds and then finalize our model.

After that we test it against the test set.

Cross Validation



Cross Validation

from sklearn.cross_validation import cross_val_score

10-fold (cv=10) cross-validation with K=5 (n_neighbors=5) for KNN (the n_neighbors parameter)

# instantiate model
knn = KNeighborsClassifier(n_neighbors=5)
# cross_val_score takes care of splitting X and y into the 10 folds that's why we pass X and y entirely
#instead of X_train and y_train
scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
print(scores)

```
[1. 0.93333333 1. 1. 0.86666667 0.93333333 0.93333333 1. 1. 1. ]
```

print(scores.mean())
0.966666666666668

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [93]:
```
# search for an optimal value of K for KNN

# list of integers 1 to 30
# integers we want to try
k_range = range(1, 31)

# list of scores from k_range
k_scores = []

# 1. we will loop through reasonable values of k
for k in k_range:
    # 2. run KNeighborsClassifier with k neighbours and obtain cross_val_score for KNeighborsClassifier with k neighbours
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=10)
    # 4. append mean of scores for k neighbors to k_scores list
    k_scores.append(scores.mean())
print(k_scores)
```

```
[0.96, 0.9533333333333334, 0.9666666666666666, 0.9666666666666666, 0.9666666666666668, 0.9666666666666668, 0.9666666666666668,
0.9666666666666668, 0.9733333333333334, 0.9666666666666668, 0.9666666666666668, 0.9733333333333334, 0.9800000000000001, 0.97333
33333333334, 0.9733333333333334, 0.9733333333333334, 0.9733333333333334, 0.9800000000000001, 0.9733333333333334, 0.980000000000
0001, 0.9666666666666666, 0.9666666666666666, 0.9733333333333334, 0.96, 0.9666666666666666, 0.96, 0.9666666666666666, 0.9533333
333333334, 0.9533333333333334, 0.9533333333333334]
```

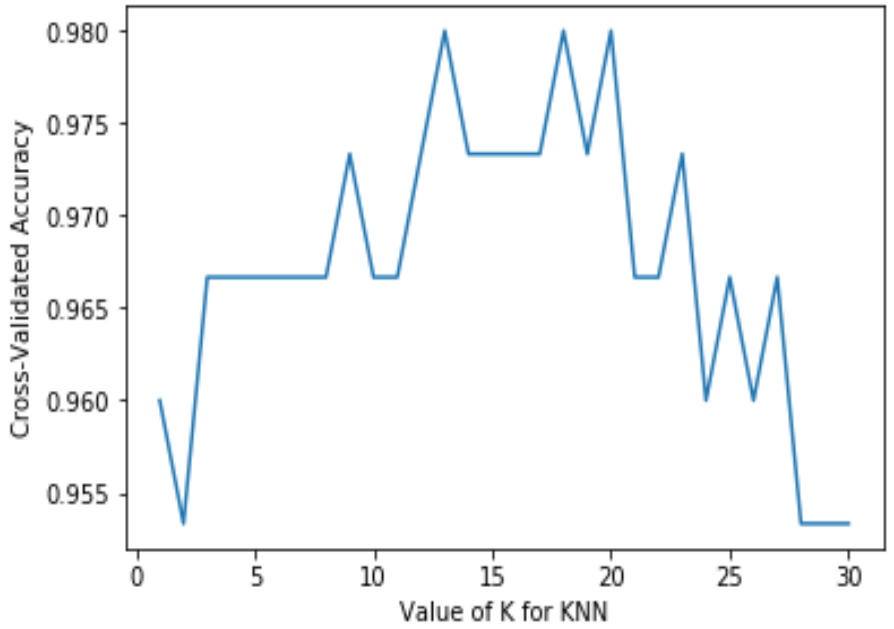
Plotting the value of K vs Cross Val. Accuracy

File Edit View Insert Cell Kernel Widgets Help

Run Code

In [71]:
```
# plot the value of K for KNN (x-axis) versus the cross-validated accuracy (y-axis)
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
```

Out[71]: Text(0,0.5,'Cross-Validated Accuracy')



The Purpose of k-fold Cross Validation

Cross-validation systematically creates and evaluates multiple models on multiple subsets of the dataset.

- This, in turn, provides a population of performance measures.
- We can calculate the mean of these measures to get an idea of how well the procedure performs on average.
- We can calculate the standard deviation of these measures to get an idea of how much the skill of the procedure is expected to vary in practice.
- Also, this information is invaluable as you can use the mean and spread to give a confidence interval on the expected performance on a machine learning procedure in practice.
- Both train-test splits and k-fold cross validation are examples of resampling methods. Resampling methods are statistical procedures for sampling a dataset and estimating an unknown quantity.

The Purpose of k-fold Cross Validation

- In the case of applied machine learning, we are interested in estimating the skill of a machine learning procedure on unseen data. More specifically, the skill of the predictions made by a machine learning procedure.
- Once we have the estimated skill, we are finished with the resampling method.
- If you are using a train-test split, that means you can discard the split datasets and the trained model.
- If you are using k-fold cross-validation, that means you can throw away all of the trained models.
- They have served their purpose and are no longer needed.
- You are now ready to finalize your model.

Train with more data?

- It won't work every time, but training with more data can help algorithms detect the signal better.
- In an example of modeling height vs. age in children, it's clear that sampling more schools will help your model.
- Of course, that's not always the case. If we just add more noisy data, this technique won't help. That's why you should always ensure your data is clean and relevant
- **Solution**
  - **Remove features: low** important features removed to improve accuracy
  - **Early stopping : Early stopping** is designed to monitor the generalization error of one model and **stop** training when generalization error begins to degrade
  - **Regularization :** techniques that help the machine to learn more than just memorize.
  - **Ensembling:** combine several **machine learning** techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking).