Meets Specifications

 Terrific job with the project! I'm impressed with how you leveraged all the main concepts of the Vehicle Detection lesson in your submission. 

To see some ideas on detecting vehicles with a deep learning approach instead, you can read [this post on the U-Net architecture](https://chatbotslife.com/small-u-net-for-vehicle-detection-9eec216f9fd6) or look into using [SSD](http://arxiv.org/abs/1512.02325) with a [Keras implementation](https://github.com/rykov8/ssd_keras).

**Writeup / README**

**The writeup / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled.**

Good work including the writeup and addressing each of the rubric items using the suggested writeup template.

**Histogram of Oriented Gradients (HOG)**

**Explanation given for methods used to extract HOG features, including which color space was chosen, which HOG parameters (orientations, pixels\_per\_cell, cells\_per\_block), and why.**

Nice job extracting the HOG features, and discussing how you arrived at your [HOG parameters](http://www.learnopencv.com/histogram-of-oriented-gradients/) by trying various combinations and using suggestions from the lesson material.

 **Tip: HOG parameters**  
To speed up the pipeline while maintaining good accuracy on the video, try using more pixels per cell along with more orientations. For example...

colorspace = 'YUV'

orient = 11

pix\_per\_cell = 16

cell\_per\_block = 2

hog\_channel = "ALL"

**The HOG features extracted from the training data have been used to train a classifier, could be SVM, Decision Tree or other. Features should be scaled to zero mean and unit variance before training the classifier.**

Good description of training the linear SVC with the [extracted HOG features](http://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/) and color histogram. 

 **Suggestions:**

* To improve the SVM's predictions on the video, you could try tuning the ["C" parameter](http://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel) — a lower C value should help with any overfitting of the training data. (e.g., try .01 or .001 instead of the default C=1)
* To reduce the feature dimensionality and help speed up the pipeline, you could also try using spatial features with size (16,16) and consider removing the color histogram — many students are able to exclude color histogram features and still get good results on the project video.
* Lastly, I've also seen students have success using an [sklearn neural net classifier](http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)...
* from sklearn.neural\_network import MLPClassifier
* clf = MLPClassifier()
* clf.fit(X\_train, y\_train)

**Sliding Window Search**

**A sliding window approach has been implemented, where overlapping tiles in each test image are classified as vehicle or non-vehicle. Some justification has been given for the particular implementation chosen.**

Nice work implementing the [sliding window search](http://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/), and describing your solution of multiple window scales.

It's generally a good idea to limit the region of interest for the search, and scan with smaller windows near the road horizon and larger windows in the bottom of the image. 

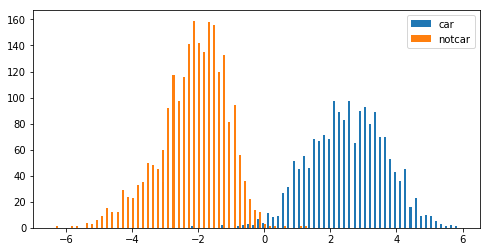
**Some discussion is given around how you improved the reliability of the classifier i.e., fewer false positives and more reliable car detections (this could be things like choice of feature vector, thresholding the decision function, hard negative mining etc.)**

Good job optimizing the performance of the classifier with your sliding window settings and classifier prediction probability threshold.

 **Tip: SVM decision function**  
To help with reducing false positives, you could also use the SVM's built-in [decision\_function](http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC.decision_function) method, which returns a confidence score based on how far a data point is from the decision boundary — higher values equate to higher confidence predictions that can be thresholded...

if svc.decision\_function(X) > threshold:

... add new detection

[](https://udacity-reviews-uploads.s3.amazonaws.com/_attachments/15447/1494516527/index.png)

**Video Implementation**

**The sliding-window search plus classifier has been used to search for and identify vehicles in the videos provided. Video output has been generated with detected vehicle positions drawn (bounding boxes, circles, cubes, etc.) on each frame of video.**

Terrific work processing the video and identifying the closest vehicles in the video!

Despite a few missed detections of the vehicles when they're farther away, it appears the classifier is doing a good job of detecting cars. 

**A method, such as requiring that a detection be found at or near the same position in several subsequent frames, (could be a heat map showing the location of repeat detections) is implemented as a means of rejecting false positives, and this demonstrably reduces the number of false positives. Same or similar method used to draw bounding boxes (or circles, cubes, etc.) around high-confidence detections where multiple overlapping detections occur.**

Nice job filtering the vehicle detections by thresholding the heatmaps, and drawing the bounding boxes after tracking detections from previous frames. 

**Discussion**

**Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.**

Good discussion of the issues you faced with the probability threshold and sliding window settings, and suggesting improvements that could be made to improve the car tracking.

* Considering that a pipeline on a real car needs to be processed in real-time, it would be great to also address the pipeline's speed performance here (e.g., describing it in frames per second).
* For additional ideas on vehicle detection, you can also read about [object tracking using opencv](https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/).