Agregar on_bad_lines=”skip”

```
# Read in word dictionary for trigrams
word_df = pd.read_csv(self.word_dict_path, names=['word'], header=None, dtype={
                                    'word': np.str}, encoding='utf-8', on_bad_lines='skip')
word_df = word_df[word_df['word'].map(lambda x: str(x).isalpha())]
word_df = word_df.applymap(lambda x: str(x).strip().lower())
word_df = word_df.dropna()
word_df = word_df.drop_duplicates()
```

Reemplazar la función as_matrix por .values

```
    not_weird = all_domains[all_domains['class'] != 'weird']
    X = not_weird[['length', 'entropy', 'alexa_grams', 'word_grams']].values

    # Labels (scikit learn uses 'y' for classification labels)
    y = np.array(not_weird['class'].tolist())
```

Reemplazar return self.clf.predict(_X)[0] por return self.clf.predict([_X])[0]

```
def predict(self, domain):
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        _alexa_match = self.alexa_counts * \
            self.alexa_vc.transform(
                [domain]).T  # Matrix multiply and transpose
        _dict_match = self.dict_counts * self.dict_vc.transform([domain]).T
        _X = [len(domain), self.entropy(
            domain), _alexa_match, _dict_match]
        if int(sklearn.__version__.split('.')[1]) > 20:
            _X = [_X]
            print(_X)
        return self.clf.predict([_X])[0]
```